Fast Computation of Special Resultants

Alin Bostan, Philippe Flajolet, Bruno Salvy, Éric Schost

▶ **To cite this version:**

Alin Bostan, Philippe Flajolet, Bruno Salvy, Éric Schost. Fast Computation of Special Resultants. Journal of Symbolic Computation, 2006, Journal of Symbolic Computation, 41 (1), pp.1-29. 10.1016/j.jsc.2005.07.001 . inria-00000960v2

HAL Id: inria-00000960

https://inria.hal.science/inria-00000960v2

Submitted on 12 Apr 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Fast Computation of Special Resultants

Alin Bostan [a] Philippe Flajolet [a] Bruno Salvy [a] Éric Schost [b]

[a] *Algorithms Project, Inria Rocquencourt, 78153 Le Chesnay, France*
[b] *LIX, École polytechnique, 91128 Palaiseau, France*

Abstract

We propose fast algorithms for computing *composed products* and *composed sums*, as well as *diamond products* of univariate polynomials. These operations correspond to special multivariate resultants, that we compute using power sums of roots of polynomials, by means of their generating series.

inria-00000960, version 2 - 12 Apr 2006

1 Introduction

Let k be a field and let f and g be monic polynomials in $k[T]$, of degrees m and n respectively. We are interested in computing efficiently their *composed sum* $f \oplus g$ and *composed product* $f \otimes g$. These are polynomials defined by

$$f \oplus g = \prod_{\alpha,\beta} \Big(T - (\alpha + \beta)\Big) \quad \text{and} \quad f \otimes g = \prod_{\alpha,\beta} (T - \alpha\beta),$$

the products running over all the roots α of f and β of g, counted with multiplicities, in an algebraic closure $\overline{k}$ of k.

More generally, given a bivariate polynomial $H \in k[X, Y]$, of degree less than m in X and less than n in Y, we study the fast computation of the *diamond product* $f \diamond_H g$, which is the polynomial of degree $D = mn$ defined by

$$f \diamond_H g = \prod_{\alpha,\beta} \Big(T - H(\alpha, \beta)\Big), \tag{1}$$

the product running over all the roots of f and g, counted with multiplicities.

Email addresses: Alin.Bostan@inria.fr (Alin Bostan), Philippe.Flajolet@inria.fr (Philippe Flajolet), Bruno.Salvy@inria.fr (Bruno Salvy), Eric.Schost@polytechnique.fr (Éric Schost).

Article published in Journal of Symbolic Computation 41 (2006) 1–35

The operation $\diamond_H$ was introduced by Brawley and Carlitz (1987). They showed the following property: if k is finite, then for large families of polynomials H, the polynomial $f \diamond_H g$ is irreducible if and only if both f and g are irreducible and their degrees are coprime. Thus, diamond products are used for constructing irreducible polynomials of large degree over finite fields, see (Brawley et al., 1999; Shoup, 1990, 1994). They occur as subroutines in many other algorithms, including computations with algebraic numbers, symbolic summation and the study of linear recurrent sequences. We present these applications in Section 5.

The polynomials $f \oplus g$ and $f \otimes g$ can be expressed by means of bivariate resultants, see for instance (Loos, 1983):

$$(f \oplus g)(T) = \mathrm{Res}_X(f(T-X), g(X)), \quad (f \otimes g)(T) = \mathrm{Res}_X(X^m f(T/X), g(X)). \tag{2}$$

A similar formula also holds for the diamond product $f \diamond_H g$:

$$(f \diamond_H g)(T) = \mathrm{Res}_X\Big(\mathrm{Res}_Y\big(T - H(X,Y), f(Y)\big), g(X)\Big). \tag{3}$$

Formulae (2) and (3) show that $f \otimes g$, $f \oplus g$ and $f \diamond_H g$ have coefficients in k. They also provide a way of computing these polynomials. Still, the complexity of the resulting algorithms is not satisfactory. For instance, if f and g have degrees of order $\sqrt{D}$, the fastest existing algorithms for bivariate resultants (Schwartz, 1980; Lickteig and Roy, 1996; Reischert, 1997; Lickteig and Roy, 2001) based on Formulae (2) have complexity of order $\widetilde{O}(D\,\mathsf{M}(\sqrt{D}))$ field operations, while the one exploiting Formula (3) has complexity $\widetilde{O}(D^2\,\mathsf{M}(\sqrt{D}))$.

In this article the symbol $\widetilde{O}$ indicates the omission of logarithmic terms, while $\mathsf{M}(d)$ stands for the number of operations in k required to perform the product of two polynomials of degree at most d. To prove complexity estimates, we implicitly use the inequality $\mathsf{M}(d_1) + \mathsf{M}(d_2) \leq \mathsf{M}(d_1 + d_2)$ for all positive integers d_1, d_2. Using algorithms based on Fast Fourier Transform (Schönhage and Strassen, 1971; Schönhage, 1977; Cantor and Kaltofen, 1991), see also (von zur Gathen and Gerhard, 1999, Section 8.2), $\mathsf{M}(d)$ can be taken in $\widetilde{O}(d)$. We also use a function denoted by $\mathsf{M}(n,d)$ which represents the complexity of multiplication of two power series in n variables and truncated at order d with respect to each variable. By a recent algorithm by Schost (2005), $\mathsf{M}(n,d)$ is in $O(n^2\mathsf{M}(d)d^n \log(d))$. Using FFT yields $\mathsf{M}(n,d) = \widetilde{O}(n^2 d^{n+1})$, which is almost optimal with respect to the size d^n of the support of the power series.

Over fields of characteristic zero, an algorithm for computing composed sums and products was given by Dvornicich and Traverso (1989). The key idea is to represent polynomials by the power sums of their roots. We will call this the *Newton representation*. Dvornicich and Traverso gave formulae expressing $f \oplus g$ and $f \otimes g$ in terms of f and g in Newton representation. However, a direct application of their formulae, combined with the use of Newton formulae for

conversions between Newton representation and the monomial one, lead to algorithms of complexity $O(D^2)$ (which is slower than the resultant method).

Brawley et al. (1999) proposed several algorithmic solutions for the composed product and sum over a finite field. Apart from the resultant method described above, their most efficient solution has quadratic complexity in the degree D of the output and works only under the assumption of an irreducible output. They also considered the problem of computing the diamond product. Their algorithm works over a finite field with q elements and has complexity $\widetilde{O}(D\log(q) + D^3)$, if f and g have degrees of order $\sqrt{D}$.

Our contribution

Our aim is to show that a better complexity can be achieved, in any characteristic. One of the keys of our approach is the use of fast algorithms due to Schönhage (1982) and Pan (2000) for converting a polynomial from the classical monomial representation to its Newton representation, and backwards.

Another crucial ingredient is our reformulation, in terms of generating series, of some formulae by Dvornicich and Traverso (1989) expressing $f \otimes g$ and $f \oplus g$ in their Newton representation. This approach enables us to give *nearly optimal* algorithms for the composed product and the composed sum, provided the characteristic of k is zero or large enough. Our algorithms use mainly multiplications, inversions and exponentiations of power series, for which nearly optimal algorithms are known (Sieveking, 1972; Kung, 1974; Brent, 1976), see also (Henrici, 1986, Section 13.9; Bini and Pan, 1994; Bürgisser et al., 1997, Chapter 2; von zur Gathen and Gerhard, 1999, Section 9.1). Throughout this article, "nearly optimal" means that the number of operations in k is linear in the size of the result, up to logarithmic factors.

Our algorithm for the composed product can be slightly modified so as to work in small characteristic as well, but the situation is different for the composed sum. By introducing a new combinatorial idea, we reduce the computation of composed sums in small characteristic p to the multiplication of two multivariate power series at order less than p in each variable. Combined with the algorithm given by Schost (2005) for multiplying multivariate power series with respect to partial truncation, this yields a nearly optimal algorithm for composed sums in small characteristic.

We also propose a fast algorithm for computing the diamond product. The heart of our method consists in relating the Newton representation of $f \diamond_H g$ to the traces of multiplication by successive powers of H in the quotient algebra $Q = k[X,Y]/(f(X), g(Y))$. This way, the computation of $f \diamond_H g$ mainly reduces to solving the *power projection problem* in Q: given an element $x \in Q$, compute the sequence $\ell(1), \ell(x), \ldots, \ell(x^N)$, where ℓ is a linear form on Q and $N \geq 1$.

For the latter problem, we propose an algorithm using $O\left(\sqrt{D}(\mathsf{M}(D) + D^{\omega/2})\right)$ operations in k. In this article ω denotes a feasible exponent of matrix multiplication over the field k, that is, a positive real number such that any two $n \times n$ matrices over k can be multiplied using $O(n^\omega)$ operations in k.

The same complexity result for the bivariate power projection has already been given (in a slightly more general context) by Shoup (1994); see also (Kaltofen, 2000). Shoup's result is an existence theorem; we build upon his idea and we exhibit an explicit algorithm with this complexity. We refer to Section 4 for historical notes on this subject.

Combining our algorithm for the power projection problem in the quotient $k[X,Y]/(f(X), g(Y))$ with the fast conversion techniques mentioned above we obtain an algorithm for the diamond product whose complexity is in $O\left(\sqrt{D}(\mathsf{M}(D) + D^{\omega/2})\right)$. Plugging the best upper bound known to this date $\omega \approx 2.376$ by Coppersmith and Winograd (1990), and using Fast Fourier Transform for the power series multiplication, we infer that the complexity of our algorithm is in $O(D^{1.688})$ [1] . For the time being, this complexity result has only a theoretical relevance, since the algorithm corresponding to $\omega \approx 2.376$ is of no practical use. In contrast, using Strassen's (1969) algorithm for matrix multiplication, with exponent $\log_2(7) \simeq 2.807$, yields a practical $O(D^{1.904})$ algorithm for the diamond product, whose experimental success is reported in Section 4. Even using naive matrix multiplication ($\omega = 3$), our algorithm is faster than previously known algorithms roughly by a factor of $\sqrt{D}$.

We encapsulate our complexity results in the following theorem. Our algorithms for the composed sums and products and for the diamond product work under no additional assumption if the base field has characteristic zero or large enough. Over fields of small positive characteristic, they require some mild assumptions, which are satisfied, for instance, if the output is squarefree. If these assumptions are not satisfied, then only a divisor of the correct result is returned.

Theorem 1 *Let k be a field of characteristic p and let f and g be two monic polynomials in $k[T]$ of degrees m and n. Let $D = mn$.*

(1) If $p = 0$ or $p > D$, then the composed operations $f \otimes g$ and $f \oplus g$ can be performed using $O(\mathsf{M}(D))$ operations in k.
(2) If $p < D$ is larger than all the multiplicities of the roots of $f \otimes g$, then $f \otimes g$ can be computed in $O\left(p\, \mathsf{M}\left(\frac{D}{p}\right) \log\left(\frac{D}{p}\right) + \mathsf{M}(D)\right)$ operations in k.
(3) If $p < D$ is larger than all the multiplicities of the roots of $f \oplus g$, then $f \oplus g$ can be computed in $O\left(p\, \mathsf{M}\left(\frac{D}{p}\right) \log\left(\frac{D}{p}\right) + \mathsf{M}\left(\frac{\log(D)}{\log(p)}, p\right)\right)$ operations in k.

[1] The exponent 1.688 may be slightly improved to 1.667 by using the faster algorithms for rectangular matrix multiplication by Huang and Pan (1998).

Let $H \in k[X,Y]$ have degree less than m in X and degree less than n in Y.

(4) If p is zero or larger than all the multiplicities of the roots of $f \diamond_H g$, then $f \diamond_H g$ can be computed in $O\left(\sqrt{D}\Big(\mathsf{M}(D) + D^{\omega/2}\Big)\right)$ operations in k.

Taking $\mathsf{M}(D) \in \widetilde{O}(D)$ and $\mathsf{M}(n,d) \in \widetilde{O}(n^2 d^{n+1})$ justifies our claims on the near optimality of our algorithms for the composed product and sum.

Outline of the article

In Section 2, we recall known fast algorithms for the translation between classical and Newton representation of univariate polynomials. In Section 3 we use these results to compute the composed product and sum, and we present the experimental behavior of the resulting algorithms. In Section 4 we study the fast computation of the diamond product $f \diamond_H g$ and provide experimental results. Section 5 presents applications of composed operations and describes two related questions: computation of resolvents and Graeffe polynomials.

Notation

- $N_s(h)$ denotes the sth *power sum* of the roots of a polynomial $h \in k[T]$, *i.e.*, the sum $\sum_\gamma \gamma^s$, taken over all the roots of h in $\overline{k}$, counted with multiplicities.
- The *Newton series* of h is the power series $\mathrm{Newton}(h) = \sum_{s \geq 0} N_s(h) T^s$.
- If P is a polynomial in $k[T]$ of degree at most n, we write $\mathrm{rev}(n, P)$ for its nth *reversal*, namely for the polynomial $P\left(\frac{1}{T}\right)T^n$.
- For $h > l \geq 0$, we use the operations $\lceil . \rceil^h$, $\lfloor . \rfloor_l$ and $[.]_l^h$ on $P = \sum_{i=0}^n p_i T^i$:

$$\lceil P \rceil^h = \sum_{i=0}^{h-1} p_i T^i, \quad \lfloor P \rfloor_l = \sum_{i=0}^{n-l} p_{i+l} T^i, \quad [P]_l^h = \sum_{i=0}^{h-l-1} p_{i+l} T^i.$$

- Given a power series $S = \sum_{i \geq 0} s_i T^i \in k[[T]]$ and an integer $m \geq 1$, we write $S \mod T^m$ for the truncated power series $\sum_{i \geq 0}^{m-1} s_i T^i$.
- $\lfloor x \rfloor$ and $\lceil x \rceil$ respectively denote the largest integer less than or equal to x, and the smallest integer larger than or equal to x.
- For a k-vector space V, we denote by $\widehat{V}$ its dual, that is, the k-vector space of k-linear maps $\ell : V \to k$.

2 Fast Conversion between Polynomials and Power Sums

As mentioned in the introduction, the speed-up that we obtain in computing composed and diamond operations is based on the use of an alternative encod-

ing for univariate polynomials, the *Newton representation* by power sums of roots. The use of the Newton representation for polynomials is classical. It is already present in (Le Verrier, 1840), and also in (Lascoux (1986); Dvornicich and Traverso (1989); Valibouze (1989); Giusti et al. (1989); Thiong Ly (1989); Schönhage (1993); González-Vega and Trujillo (1995a; 1995b); González-López and González-Vega (1998); Rouillier (1999); van Hoeij (2002); Briand and González-Vega (2002)). Pushing further an idea of (Dvornicich and Traverso, 1989), we show that Newton representation provides the appropriate data structure for the efficient computation of composed and diamond operations.

In characteristic zero or larger than D, any polynomial of degree D is uniquely determined by the first D power sums of its roots. Newton formulae provide a straightforward algorithm to perform these conversions, but its complexity is quadratic in D. Fortunately, faster conversion methods exist. We thus recall algorithms due to Schönhage and Pan in this section; they will be used as basic algorithmic bricks in the rest of our article. For the sake of completeness, we collected them under the shape of ready-to-implement pseudo-code.

The structure of this section is as follows: we begin by recalling an algorithm for the direct conversion (from a polynomial to its Newton representation), which works in arbitrary characteristic. Next, we deal with the inverse conversion in characteristic zero or large enough. We conclude the section by presenting an algorithm for the inverse conversion in the positive characteristic setting.

2.1 From monomial to Newton representation

Schönhage (1982) was the first to propose an algorithm for the translation from monomial to Newton representation. It is based on the following result.

Lemma 1 *Let h be a monic polynomial in $k[T]$, of degree D. Then, the series* $\mathrm{Newton}(h)$ *is rational; moreover, the following formula holds:*

$$\mathrm{Newton}(h) = \frac{\mathrm{rev}(D-1, h')}{\mathrm{rev}(D, h)}.$$

Proof. Let $\gamma_1, \ldots, \gamma_D$ be the roots of h in $\overline{k}$. Since $h = \prod_{i=1}^{D}(T - \gamma_i)$, we have

$$\mathrm{Newton}(h) = \sum_{s \geq 0} \Big(\sum_{i=1}^{D} \gamma_i^s \Big) T^s = \sum_{i=1}^{D} \Big(\sum_{s \geq 0} \gamma_i^s T^s \Big) = \sum_{i=1}^{D} \frac{1}{1 - \gamma_i T} = \frac{\mathrm{rev}(D-1, h')}{\mathrm{rev}(D, h)}.$$

□

Proposition 1 *If $h \in k[T]$ has degree D and if $N \geq D$, then the first N power sums of the roots of h can be computed in $O\Big(\frac{N}{D}\mathsf{M}(D)\Big)$ operations in k.*

Proof. By Lemma 1, it is sufficient to prove that if a polynomial A has degree at most $D-1$ and if a polynomial B has degree D, then the first $N \geq D$ coefficients of the rational series A/B can be computed within the announced running time bound. The idea is to proceed by *slices* of size D. We first compute the first D coefficients of $1/B$, using the Sieveking-Kung algorithm (Sieveking, 1972; Kung, 1974), for a cost of $O(\mathsf{M}(D))$ operations in k. Denote B_0 the corresponding polynomial, of degree $D-1$. We let $C_0 = \lceil AB_0 \rceil^D$ and recursively define the polynomials

$$C_{j+1} = -\left\lceil \lfloor BC_j \rfloor_D B_0 \right\rceil^D, \quad \text{for } 0 \leq j \leq \lfloor N/D \rfloor .$$

Then, it is easy to check that $\frac{A}{B} = C_0 + T^D C_1 + T^{2D} C_2 + \dots$ and the result follows. □

The corresponding algorithm is summarized in Figure 1.

Input: a polynomial h of degree D.
Output: Newton(h) at precision $N \geq D$.

$A \leftarrow \mathrm{rev}(D-1, h')$
$B \leftarrow \mathrm{rev}(D, h)$
$B_0 \leftarrow \left\lceil \frac{1}{B} \right\rceil^D$
$C_0 \leftarrow \lceil AB_0 \rceil^D$
$l \leftarrow \left\lfloor \frac{N}{D} \right\rfloor$
`for` j `from` 0 `to` l `do`
    $C_{j+1} \leftarrow -\left\lceil \lfloor BC_j \rfloor_D B_0 \right\rceil^D$
`return` $\sum_{i=0}^{l} C_i T^{Di} + O(T^N)$

Fig. 1. Computing the Newton series of a polynomial

2.2 From Newton representation to monomial representation

The converse direction is more difficult to handle: while in characteristic zero the Newton formulae give a one-to-one correspondence between power sums and elementary symmetric polynomials, in the positive characteristic case distinct monic polynomials of the same degree may have equal power sums of roots (*e.g.*, T^2 and T^2+1 over $\mathbb{F}_2$). Consequently, the treatment of this question should take into account the characteristic of the base field. Many efforts have been made to bypass this difficulty, see Subsection 2.2.2 for historical

details. The content of the next subsections is encapsulated in the following proposition.

Proposition 2 (Schönhage (1982); Pan (2000)) *Let h be a polynomial of degree D in $k[T]$.*

(1) If k has characteristic zero or greater than D, then h can be computed from the first D power sums of its roots within $O(\mathsf{M}(D))$ operations in k.
(2) Suppose that k has positive characteristic p and that all the roots of h have multiplicities less than p. Then, the number of operations in k needed to compute the polynomial h from the first $2D$ power sums of its roots is

$$O\left(\mathsf{M}(D) + p\,\mathsf{M}\Big(\frac{D}{p}\Big)\log\Big(\frac{D}{p}\Big)\right).$$

In Subsection 2.2.1 we treat the case of characteristic zero or large enough, since the ideas involved in that case are important and help understand the extension to the arbitrary positive characteristic case. The latter case is addressed in Subsection 2.2.2, where technical details are intentionally omitted. Instead we preferred to write down complete pseudo-code, to simplify the task of reading Pan's original article (Pan, 2000).

2.2.1 The case of characteristic zero or large enough

The exponential of a power series F of positive valuation over a field k is given by

$$\exp(F) = \begin{cases} \sum_{s\geq 0} F^s/s!, & \text{if } \operatorname{char}(k) = 0, \\ \sum_{s=0}^{p-1} F^s/s!, & \text{if } \operatorname{char}(k) = p > 0. \end{cases}$$

The next result is a converse of Lemma 1.

Lemma 2 *Let h be a monic polynomial of degree D in $k[T]$, where k is a field of characteristic zero or larger than D. Then the following formula holds:*

$$\operatorname{rev}(D,h) = \exp\left(\int \frac{1}{T}\cdot\Big(D - \operatorname{Newton}(h)\Big)\right).$$

Proof. Let $\gamma_1,\ldots,\gamma_D$ be the roots of h in $\overline{k}$. By Lemma 1, it follows that:

$$\frac{\operatorname{rev}(D,h)'}{\operatorname{rev}(D,h)} = \sum_i \frac{-\gamma_i}{1-\gamma_i T} = -\sum_{s\geq 0}\Big(\sum_i \gamma_i^{s+1}\Big)T^s \tag{4}$$

and, by definition, the right-hand side equals $\Big(D - \operatorname{Newton}(h)\Big)/T$.

As one can easily verify, for $P \in k[T]$ with constant coefficient 1, the formula $P = \exp(\int P'/P)$ holds as soon as k has characteristic zero. The same equality remains valid modulo T^p if $p = \mathrm{char}(k)$ is larger than the degree of P. By applying this fact to $P = \mathrm{rev}(D, h)$, we conclude the proof of the lemma. □

Corollary 1 *A monic polynomial h of degree D over a field of characteristic zero or larger than D can be computed from the first D power sums of its roots within $O(\mathsf{M}(D))$ base field operations.*

Proof. The primitive of the power series $(D - \mathrm{Newton}(h))/T$ can be computed at precision D in linear time. By Lemma 2, exponentiating the latter series gives the polynomial $\mathrm{rev}(D, h)$. This exponential can be computed within $O(\mathsf{M}(D))$ field operations, (Brent, 1976). Finally, we recover the polynomial $h = \mathrm{rev}\Big(D, \mathrm{rev}(D, h)\Big)$. □

The first part of Proposition 2 is now proved. The resulting algorithm is presented in Figure 2. For a polynomial P, we denote by $\texttt{Coeff}(P, i)$ the coefficient of T^i in P. We use a variant of the algorithm in (Brent, 1976) for power series exponential based on Newton iteration, but whose complexity has a better constant factor and which is similar to that of (Pan, 1997, Appendix A).

Input: the first D terms of $\mathrm{Newton}(h)$.
Output: the polynomial h.

$S \leftarrow (\mathrm{Newton}(h) - D)/T$
$R \leftarrow 1 - \texttt{Coeff}(S, 0)T$
$n \leftarrow 2$
`while` $n \leq D$ `do`
  $M' \leftarrow -\left\lceil \frac{R'}{R} + S \right\rceil^{2n-1}$
  $M \leftarrow 1 + \sum_i \texttt{Coeff}(M', i)\frac{T^i}{i}$
  $R \leftarrow \lceil RM \rceil^{2n}$
  $n \leftarrow 2n$
$R \leftarrow \lceil R \rceil^{D+1}$
`return` $\mathrm{rev}(D, R)$

Fig. 2. Recovering a polynomial from its Newton series in characteristic zero

2.2.2 The small positive characteristic case – the Schönhage-Pan algorithm

The conversion from Newton sums to coefficients in small characteristic is a more subtle problem. Historically, two kinds of approaches have been pro-

posed [2] : on the one hand, the techniques of *recursive triangulation* originated in (Kaltofen and Pan, 1992, 1994), on the other hand, those using *fundamental sets of power sums* of (Schönhage, 1993; Pan, 1996, 2000). The best currently known solution is that of Pan (2000), which inherits ideas from Schönhage (1993) and from the algorithmic improvements in (Pan, 1996, 1997). The first approach was designed for parallel computations and its sequential complexity is not competitive with that of the second one; we thus focus on the latter.

Let h be a polynomial of degree D over a field of characteristic $0 < p < D$. To recover h from its first $2D$ power sums, Schönhage (1993) proposed the following method:

(1) Compute a polynomial $g = T^d + g_1 T^{d-1} + \cdots + g_d$ of degree $d > 2D$ whose first d power sums equal those of h and such that $g_{ip} = 0$ for $i \geq 1$.
(2) Starting from g, recover the polynomial h.

Intuitively, the polynomial g is obtained by applying to h those Newton formulae which do not involve division by p and setting the other coefficients to zero. This can be performed using $O(D^2)$ operations in k. In (Schönhage, 1993) this stage was completed using a reduction to a triangular linear system of size $\lfloor D/p \rfloor$, for a cost of $O\Big(\mathsf{M}(D) + (D/p)^2\Big)$ operations in k. Subsequently, Pan (1996) gave an improved solution for the computation of g, by adapting Newton's iteration in the algorithm of Corollary 1 to the positive characteristic case, leading to a cost of $O\left(\mathsf{M}(D)\right)$ operations in k for this first stage.

Concerning the second stage, Schönhage (1993) proposed a solution based on the resolution of a linear system of equations of size $\lfloor D/p \rfloor$. This step was accelerated by Pan (1997), who showed that it amounts to solving $p-1$ Padé approximation problems of sizes at most $(D/p, D/p)$.

For technical details, we refer to the original articles (Schönhage, 1993; Pan, 1996, 1997, 2000). We give in Figure 3 the algorithm we extracted from (Pan, 2000). This algorithm takes as input the first $2D$ terms of the series Newton(h) and returns the polynomial h. We use the notation $\texttt{lcm}(f_i)$ for the least common multiple of a family of polynomials (f_i) and $\texttt{Pade}(S, a, b)$ for the Padé approximant (A, B) of a power series (polynomial) S, that is the (unique) pair of polynomials A and B of minimal degrees such that $B(0) = 1$ and such that the following holds:

$$A - BS = 0 \mod T^{a+b+1}, \ \deg(A) \leq a, \ \deg(B) \leq b.$$

A conceptually simpler algorithm was given in (Bostan et al., 2002, Lemma 3).

[2] A third approach, specific to finite fields, has been recently developed by Bostan et al. (2005), but the input required by that algorithm is different.

The Schönhage-Pan algorithm

Input: the first $2D$ terms of Newton(h).
Output: the polynomial h.

$S \leftarrow (\text{Newton}(h) - D)/T$
$R \leftarrow 1 - \texttt{Coeff}(S, 0)T$
$n \leftarrow 2$
`while` $n \leq 2D$ `do`
  $M' \leftarrow -\left\lceil \frac{R'}{R} + S \right\rceil^{2n-1}$
  $M \leftarrow 1 + \sum_{p \nmid i} \texttt{Coeff}(M', i)\frac{T^i}{i}$
  $g \leftarrow \lfloor \frac{2n-1}{p} \rfloor$
  `if` $gp > n$ `then`
    $A \leftarrow [R]_1^{gp-n+1}\,[M]_n^{gp}$
    $M \leftarrow M - \sum_{i=\lceil \frac{n}{p} \rceil}^{g} \texttt{Coeff}(A, ip - n - 1)T^{ip}$
  $R \leftarrow \lceil RM \rceil^{2n}$
  $n \leftarrow 2n$
`for` i `from` 1 `to` $p - 1$ `do`
  $d_i \leftarrow \left\lfloor \frac{D-i}{p} \right\rfloor$
  $Q_i \leftarrow \sum_{j=0}^{d+d_i} \texttt{Coeff}(R, jp + i)T^j$
  $(A_i, B_i) \leftarrow \texttt{Pade}(Q_i, d_i, d)$
  $H_0 \leftarrow \texttt{lcm}(B_i)$
  $H_i \leftarrow \lceil Q_i H_0 \rceil^{d_i+1}$
$H \leftarrow H_0(T^p) + \sum_{i=1}^{p-1} H_i(T^p)T^i$
`return` $\text{rev}(D, H)$

Fig. 3. Recovering a monic polynomial from its Newton series in small characteristic

It has complexity $O(\mathsf{M}(D)\log(D))$, and, for fixed p, it differs from that of the Schönhage-Pan algorithm only by a constant factor.

3 Two Useful Resultants that Can Be Computed Fast

Designing nearly optimal algorithms for general bivariate resultants is still an open problem, see (von zur Gathen and Gerhard, 1999, Research problem 11.10). The results of the preceding section enable us to give such algorithms for the particular cases of the composed product $f \otimes g$ and composed sum $f \oplus g$. Our algorithms are based on formulae expressing the Newton series of $f \otimes g$ and of $f \oplus g$ in terms of those of f and g.

3.1 Computing the composed product

Lemma 3 *Let f and g be two polynomials in $k[T]$. Then:*

$$\mathrm{Newton}(f \otimes g) = \mathrm{Newton}(f) \odot \mathrm{Newton}(g),$$

where $\odot$ denotes the Hadamard (term-wise) product of power series.

Proof. For $s \geq 0$, the sth power sum of the roots of $f \otimes g$ is $\sum_{\alpha,\beta}(\alpha\beta)^s$, the sum running over all the roots α of f and β of g. This sum can be rewritten as $\left(\sum_\alpha \alpha^s\right) \cdot \left(\sum_\beta \beta^s\right)$, which is the product of the sth power sums of the roots of f and of g. This proves that the series $\mathrm{Newton}(f \otimes g)$ is the Hadamard product of $\mathrm{Newton}(f)$ and $\mathrm{Newton}(g)$. □

As a corollary, we obtain the following algorithm for $f \otimes g$. Given two monic polynomials f and g of degrees m and n, we first compute the power series $\mathrm{Newton}(f)$ and $\mathrm{Newton}(g)$ up to precision $D = mn$. Using Proposition 1, this step requires $O\left(\frac{D}{m}\mathsf{M}(m) + \frac{D}{n}\mathsf{M}(n)\right)$ operations in k. Then we perform, at a cost linear in D, the Hadamard product of $\mathrm{Newton}(f)$ and $\mathrm{Newton}(g)$. By the preceding lemma, we thus obtain the Newton series of $f \otimes g$ at precision D. We recover the polynomial $f \otimes g$ by applying the conversion algorithms in Proposition 2. Summing up the costs of each stage proves the complexity results concerning $f \otimes g$ in the first two assertions in Theorem 1.

3.2 Computing the composed sum in characteristic zero or large enough

Let k be a field and $\mathrm{E} \in k[[T]]$ be the power series $\exp(T)$, with exp as defined in Section 2.2.1. Then our algorithm for $f \oplus g$ is based on the following lemma:

Lemma 4 *Let f and g be two polynomials in $k[T]$. Then*

(1) If the characteristic of k is zero, the following formula holds:

$$\mathrm{Newton}(f \oplus g) \ \odot \ \mathrm{E} = \Big(\mathrm{Newton}(f) \ \odot \ \mathrm{E}\Big) \cdot \Big(\mathrm{Newton}(g) \ \odot \ \mathrm{E}\Big);$$

(2) If $p > 0$ is the characteristic of k, the following formula holds:

$$\mathrm{Newton}(f \oplus g) \ \odot \ \mathrm{E} = \Big(\mathrm{Newton}(f) \ \odot \ \mathrm{E}\Big) \cdot \Big(\mathrm{Newton}(g) \ \odot \ \mathrm{E}\Big) \mod T^p.$$

Proof. We only treat the characteristic zero case; the arguments apply *mutatis mutandis* in characteristic p, since $\exp(F+G) = \exp(F)\exp(G) \bmod T^p$.

The definition of Newton series shows that

$$\mathrm{Newton}(f \oplus g) = \sum_{s \geq 0} \left(\sum_{\alpha,\beta} (\alpha + \beta)^s \right) T^s,$$

the second sum running over the roots of f and g. The conclusion now reads

$$\sum_{s \geq 0} \frac{\sum_{\alpha,\beta} (\alpha+\beta)^s}{s!} T^s = \left(\sum_{s \geq 0} \frac{\sum_{\alpha} \alpha^s}{s!} T^s \right) \cdot \left(\sum_{s \geq 0} \frac{\sum_{\beta} \beta^s}{s!} T^s \right)$$

and we are done, as the latter equality simply translates the fact that

$$\sum_{\alpha,\beta} \exp\Big((\alpha+\beta)T\Big) = \Big(\sum_{\alpha} \exp\Big(\alpha T\Big) \Big) \cdot \Big(\sum_{\beta} \exp\Big(\beta T\Big) \Big). \qquad \square$$

As a corollary, we obtain an algorithm for computing the composed sum of two monic polynomials f and g: first, compute $\mathrm{Newton}(f)$ and $\mathrm{Newton}(g)$ to precision D, perform their Hadamard product with E, then compute the product in Lemma 4 to recover $\mathrm{Newton}(f \oplus g)$ at precision D and finally convert the last Newton series to the polynomial $f \oplus g$. Using Proposition 1 and Proposition 2, this completes the proof of the first assertion in Theorem 1.

3.3 Computing the composed sum in small characteristic

In this section we treat the computation of composed sums in small characteristic. In the case of large characteristic, our solution involved an identity related to the exponential series. Because of non-invertible factorials, the definition of the exponential series is problematic in the present case. In what follows we overcome this difficulty by appealing to certain multivariate exponential generating series.

We begin by giving the intuition behind our approach on a particular case. Suppose that f and g are polynomials over a field of characteristic $p > 0$; our aim is to express the first p^2 power sums of the roots $\sum_{\alpha,\beta}(\alpha+\beta)^\ell$ in terms of $\sum_\alpha \alpha^\ell$ and $\sum_\beta \beta^\ell$. The first p of these sums can be determined using the method in the previous section, which amounts to exploiting the identity $\exp((\alpha+\beta)T) = \exp(\alpha T) \cdot \exp(\beta T)$. For ℓ between p and p^2-1, this method fails, since one is not able to divide by p in k. In contrast, if ℓ is written as $i+pj$ with i and j less than p, then the equality $(\alpha+\beta)^\ell = (\alpha+\beta)^i(\alpha^p+\beta^p)^j$ suggests the use of the bivariate identity

$$\exp\Big((\alpha+\beta)T + (\alpha+\beta)^p U\Big) = \exp(\alpha T + \alpha^p U) \cdot \exp(\beta T + \beta^p U),$$

which translates into the following equality modulo (T^p, U^p)

$$\left(\sum_{i,j=0}^{p-1}\Big(\sum_{\alpha}\alpha^{i+pj}\Big)\frac{T^iU^j}{i!j!}\right)\left(\sum_{i,j=0}^{p-1}\Big(\sum_{\beta}\beta^{i+pj}\Big)\frac{T^iU^j}{i!j!}\right)=\sum_{i,j=0}^{p-1}\Big(\sum_{\alpha,\beta}(\alpha+\beta)^{i+pj}\Big)\frac{T^iU^j}{i!j!},$$

helping us to find the first p^2 power sums of $f\oplus g$ by means of a multiplication of bivariate power series. We now formalize this idea in the general case.

Let k be a field of characteristic $p>0$. For $i\in\mathbb{N}$, we write $\mathbf{i_p}=(i_0,\ldots,i_s)$ for its p-adic expansion, that is, the (unique) sequence of integers $0\le i_\ell<p$ such that $i=i_0+i_1p+\cdots+i_sp^s$. Let $\mathbf{T}$ be an infinite set of indeterminates $(T_i)_{i\ge0}$. We define *the p-exponential* $\mathrm{E}_p\in k[[\mathbf{T}]]$ as the multivariate power series

$$\mathrm{E}_p \;=\; \sum_{i\ge0}\frac{\mathbf{T}^{\mathbf{i_p}}}{\mathbf{i_p}!},$$

where for $\mathbf{i_p}=(i_0,\ldots,i_s)$, we note $\mathbf{i_p}!=i_0!\cdots i_s!$ and $\mathbf{T}^{\mathbf{i_p}}=T_0^{i_0}\cdots T_s^{i_s}$. For $f\in k[T]$, we define the *p-Newton series of f* as the multivariate power series

$$\mathrm{Newton}_p(f) \;=\; \sum_{i\ge0}N_i(f)\mathbf{T}^{\mathbf{i_p}}.$$

By definition, in each variable, the degree of $\mathrm{Newton}_p(f)$ is smaller than p. With this notation, our algorithm for $f\oplus g$ in small characteristic is based on the following results, which generalize Lemma 4.

Lemma 5 *Let k a field of characteristic p and let $f\in k[T]$. Then:*

$$\mathrm{Newton}_p(f)\odot\mathrm{E}_p \;=\; \sum_{s\ge0}\sum_{f(\alpha)=0}\left(\prod_{\ell=0}^{s}\exp(\alpha^{p^\ell}T_\ell)\right),$$

where $\odot$ denotes the term-wise product of multivariate power series.

Proof. By definition, the left-hand side equals

$$\sum_{\substack{i\ge0\\ \mathbf{i_p}=(i_0,\ldots,i_s)}}\frac{N_i(f)}{i_0!\cdots i_s!}\,T_0^{i_0}\cdots T_s^{i_s} \;=\; \sum_{s\ge0}\sum_{f(\alpha)=0}\left(\sum_{\substack{0\le i_0<p\\ \cdots\\ 0\le i_s<p}}\frac{\alpha^{i_0+i_1p+\cdots+i_sp^s}}{i_0!\cdots i_s!}\,T_0^{i_0}\cdots T_s^{i_s}\right).$$

The lemma follows, since all the summation indices i_ℓ vary independently and

$$\sum_{0\le i_\ell<p}\frac{\alpha^{i_\ell p^\ell}}{i_\ell!}T_\ell^{i_\ell}=\exp(\alpha^{p^\ell}T_\ell).\qquad\square$$

Lemma 6 *Let k be a field of characteristic $p>0$ and let f and g be two*

monic polynomials in $k[T]$. Then the following identity holds:

$$\mathrm{Newton}_p(f \oplus g) \odot \mathrm{E}_p \;=\; \Big(\mathrm{Newton}_p(f) \odot \mathrm{E}_p\Big) \cdot \Big(\mathrm{Newton}_p(g) \odot \mathrm{E}_p\Big) \mod (\mathbf{T}^p),$$

where $(\mathbf{T}^p)$ denotes the ideal generated by the monomials $T_0^p, \ldots T_s^p, \ldots$ in $k[[\mathbf{T}]]$.

Proof. By Lemma 5, we can rewrite the left-hand side as

$$\sum_{s\geq 0} \sum_{\substack{f(\alpha)=0\\ g(\beta)=0}} \exp\Big((\alpha+\beta)T_0\Big) \cdots \exp\Big((\alpha+\beta)^{p^s} T_s\Big)$$

and the right-hand side as

$$\sum_{s\geq 0} \sum_{\substack{f(\alpha)=0\\ g(\beta)=0}} \exp(\alpha T_0) \exp(\beta T_0) \cdots \exp(\alpha^{p^s} T_s) \exp(\beta^{p^s} T_s).$$

The conclusion of the lemma follows, using the fact that the series

$$\exp\Big((\alpha+\beta)^{p^\ell} T_\ell\Big) \;=\; \exp\Big(\alpha^{p^\ell} T_\ell + \beta^{p^\ell} T_\ell\Big)$$

is equal to $\exp(\alpha^{p^\ell} T_\ell) \exp(\beta^{p^\ell} T_\ell)$ modulo T_ℓ^p for any $\ell \geq 0$. □

Via the fast conversion algorithms in Section 2, the preceding lemma enables us to reduce the computation of the composed sum of characteristic $p > 0$ to a single multiplication of multivariate series involving a *finite number of variables* and of degree less than p in each variable. Precisely, to recover the polynomial $f \oplus g$, only $2D$ terms of its Newton series suffice, where $D = \deg(f \oplus g)$, and this means that in the p-Newton series of $f \oplus g$, all we need to know are coefficients of the monomials containing $T_0, \ldots, T_s$, where $s = \lfloor \log(2D)/\log(p) \rfloor$. By Lemma 6, this can be done by multiplying two multivariate power series in at most $\log(2D)/\log(p)$ variables and of degree less than p in each variable. This completes the proof of part (3) in Theorem 1.

Using the algorithm of Schost (2005), the last multivariate multiplication of power series can be performed in $\widetilde{O}(pD)$. A simpler (but slower) alternative algorithm relies on Kronecker's substitution, see (Kronecker, 1882, §4) and (von zur Gathen and Gerhard, 1999, Section 8.4); its cost is $\widetilde{O}(D^{1+\frac{1}{\log(p)}})$.

3.4 Experimental results

We have implemented the algorithms for the composed sum and product, in their versions for large or zero characteristic. We used the NTL C++ library as a basis (Shoup, 1996–2005).

Since our complexity estimates are stated in terms of the number of operations executed in the base field, we have chosen to experiment on finite fields of the form $\mathbb{Z}/p\mathbb{Z}$, with p prime. For such fields, NTL implements polynomial arithmetic using classical, Karatsuba and Fast Fourier Transform multiplications.

For the tests [3] presented in Figure 4, the input polynomials have equal degrees $m = n$ and their coefficients are chosen uniformly at random; the output has degree $D = m^2$. We let m vary from 1 to 500 by steps of 8, so that D varies from 1 to 250000; the base field is defined by a 32 bit prime. We stress the fact that such output degrees are met in applications, see Section 5.

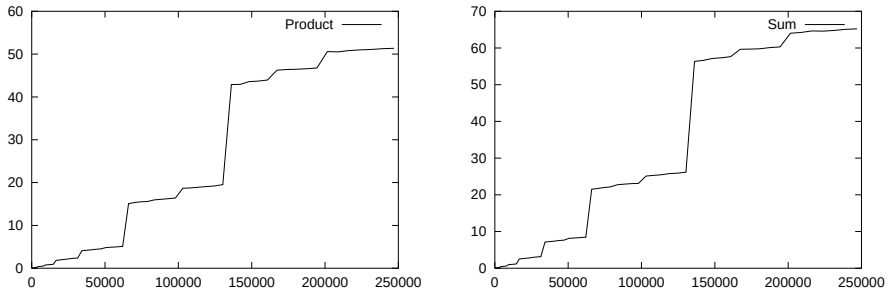

Fig. 4. Composed product and sum. (Time in sec. vs output degree)

In both cases (composed product and composed sum), the running times present an abrupt jump when the output degree D passes a power of 2. This feature is actually already present in polynomial multiplication, and is inherent in NTL's use of the Fast Fourier Transform: see Figure 5, which displays the time for one polynomial multiplication, in the same degree range as Figure 4. We also plot the ratios between the times of composed sum (resp. product) and polynomial multiplication. For large degrees, the ratios do not exceed 5.

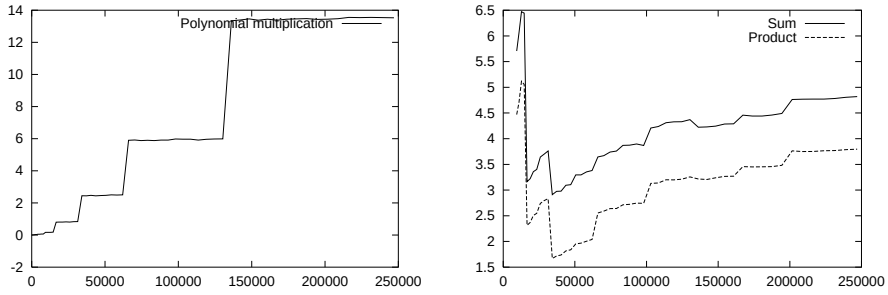

Fig. 5. Left: polynomial multiplication (Time in sec. vs output degree). Right: (Composed product or sum time) / (Multiplication time) vs output degree.

We also implemented in NTL an algorithm based on Formulae (2), as well

[3] All our tests were performed on the computers of the MEDICIS resource center http://www.medicis.polytechnique.fr, using a 2 GB, 2200+ AMD Athlon processor.

as the algorithm by Dvornicich and Traverso (1989). The bivariate resultant computation relies on NTL's built-in implementation of both quadratic and fast algorithms for univariate resultants. The latter is used for m larger than 180, i.e. D larger than 32400. The experimental timings are given in Figure 6. A first observation is that, in accordance with theoretical estimates, the algorithm of Dvornicich and Traverso (1989) is slower than the resultant based algorithm. The main conclusion is that these experimental results show that for large degrees, the resultant computations take several hours, whereas our algorithms require approximately one minute.

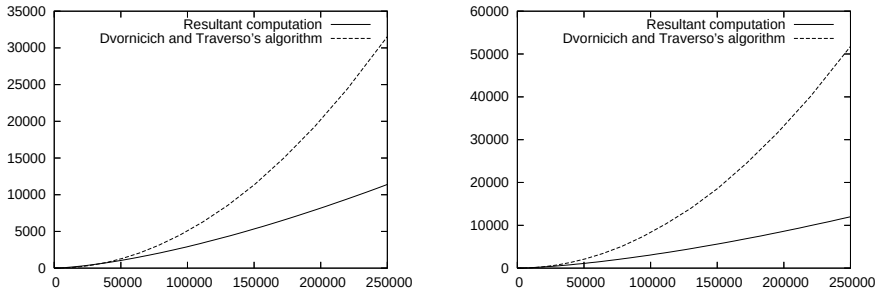

Fig. 6. Composed product (left) and sum (right) by resultant computation, respectively by Dvornicich and Traverso (1989) algorithm. (Time in sec. vs output degree)

4 Computing the Diamond Product

We finally address the general case: computing the *diamond product* of f and g. In this section, we give an algorithm that computes $f \diamond_H g$ using $O\left(\sqrt{D}\left(\mathsf{M}(D) + D^{\omega/2}\right)\right)$ operations in k. Anticipating the following section, $f \diamond_H g$ is the characteristic polynomial of the image of H in the quotient algebra $k[X,Y]/(f(X), g(Y))$. This characterization of the diamond product enables us to reduce its computation to that of the *power projections* of H under the *trace* map, followed by a linearly generated sequence recovery.

The origins of this approach can be traced back at least to Le Verrier (1840) who proposed a method for computing characteristic polynomials of matrices by means of traces of matrix powers and using Newton identities for the recovery step. The idea of using power projections for computing minimal polynomials in quotient algebras appears in (Thiong Ly, 1989; Rifà and Borrell, 1991) in the one variable case $k[X]/(f(X))$. A breakthrough was achieved by Shoup (1994), who was the first to notice that the power projection problem is *dual* to the *modular composition problem*.

Combining a complexity result of Brent and Kung (1978) for the latter problem with an algorithmic theorem called *Tellegen's principle*, Shoup proved the existence of an algorithm solving the power projection problem within the same

complexity as ours, even for the more general algebra $k[X,Y]/(f(X),g(X,Y))$. Still, Shoup gave no explicit algorithm. In (Shoup, 1995, Section 4.1 and 7.5) he partially filled in this gap and proposed a "baby step/giant step" algorithm for the power projection in the univariate case, in the FFT polynomial multiplication setting; in a subsequent article (Shoup, 1999, Section 2.2), he extended his algorithm to the bivariate case, and independently of the polynomial multiplication model. Yet, in the complexity of algorithms in (Shoup, 1995, 1999), the term $D^{\omega/2}$ is replaced by $D^{3/2}$. Finally, Kaltofen (2000) describes the bivariate power projection and its relation to modular composition.

In this section, we follow the steps of Kaltofen and Shoup and we solve the power projection problem for the quotient algebra $Q = k[X,Y]/(f(X),g(Y))$ within the complexity predicted by Tellegen's principle. This is done by applying effective transposition techniques by Bostan et al. (2003a) to Brent and Kung's algorithm for modular composition in Q. We refer to Bostan et al. (2003b) for a description of the general multivariate power projection problem, and its applications to the context of polynomial system solving.

4.1 Computations in the quotient algebra

Let Q be the quotient algebra $k[X,Y]/(f(X),g(Y))$. In the rest of this section, we repeatedly use the *trace*, which is a linear form defined on Q: the trace of $A \in Q$ is defined as the trace of the map of multiplication by A in Q.

Our algorithm for the diamond product is based on the following fundamental fact, which is sometimes referred to as Stickelberger's Theorem, see (Cox et al., 1998, Proposition 2.7): for any A in Q, the characteristic polynomial of A equals $\prod_{\alpha,\beta}\Big(T - A(\alpha,\beta)\Big)$, where the product runs over all the roots of f and g counted with multiplicities. As a corollary, we have the following:

Lemma 7 *The polynomial $f \diamond_H g$ is the characteristic polynomial of H in Q. The sth power sum of the roots of $f \diamond_H g$ is the trace of H^s in Q.*

The second part of Lemma 7 together with the fast conversion algorithms of Section 2 show that the proof of the final part of Theorem 1 amounts to giving a fast computation scheme for the first traces of H^s in Q. This is the object of the following proposition.

Proposition 3 *Given $N \geq 1$, the sequence*

$$\mathrm{trace}(1), \mathrm{trace}(H), \mathrm{trace}(H^2), \ldots, \mathrm{trace}(H^{N-1})$$

can be computed within $O\Big(\sqrt{N}\,\mathsf{M}(D) + DN^{(\omega-1)/2}\Big)$ base field operations.

The proof of the complexity estimates in Theorem 1 follows directly: from Proposition 2, the number of traces to be computed is at most $2D$ and by Proposition 3, this has complexity $O\Big(\sqrt{D}(\mathsf{M}(D)+D^{\omega/2})\Big)$. Then Proposition 2 and the obvious inequality $p\,\mathsf{M}(\frac{D}{p})\log(\frac{D}{p}) \leq \mathsf{M}(D)\log(D)$ show that the cost of recovering $f \diamond_H g$ from the power sums of its roots is negligible. Thus, we now concentrate on proving Proposition 3.

4.2 Power projection

Computing the traces of the first N powers of H is a particular instance of the *power projection* problem: given a linear form ℓ on the k-algebra Q, compute the image of ℓ under the linear map

$$\begin{aligned} \widehat{Q} &\to k[[T]]_{<N} \\ \ell &\mapsto \sum_{i=0}^{N-1} \ell(H^i)T^i + O(T^N). \end{aligned}$$

It is useful to notice that $k[[T]]_{<N}$ naturally identifies, as a k-vector space, with the dual of the space $k[T]_{<N}$ formed by polynomials of degree at most $N-1$. Under this identification, the linear map defining the power projection is the transposed map of the *modular composition* (*polynomial evaluation*) by H

$$\begin{aligned} k[T]_{<N} &\to Q \\ p &\mapsto p(H). \end{aligned}$$

Now, an algorithmic theorem called *transposition principle*, or *Tellegen's principle*, states, roughly speaking, that for any algorithm computing a linear map there exists an algorithm that computes the transposed map using almost the same number of arithmetic operations (see the next proposition for a precise statement). In our situation, this principle establishes a computational equivalence between the dual problems of modular composition and power projection. This observation is due to Shoup (1994); see also (Kaltofen, 2000).

Tellegen's principle can be phrased in terms of *linear straight-line programs*; these are "ordinary" straight-line programs, that use only linear operations, see Chapter 13 in (Bürgisser et al., 1997) for precise definitions. The complexity of a linear straight-line program is measured by its size, that is, the number of operations it uses.

Proposition 4 (Bürgisser et al., 1997, Th. 13.20) *Let* $\mathbf{M}$ *be a* $m \times n$ *matrix with* z_r *zero rows and* z_c *zero columns. For every linear straight-line program of size* L *that computes the matrix-vector product* $\mathbf{Mv}$ *there exists a linear*

straight-line program of size $L - n + m - z_r + z_c$ *that computes the transposed matrix-vector product* $\mathbf{M}^{\mathrm{tr}}\mathbf{v}$.

Paterson and Stockmeyer (1973) proposed a "baby-step / giant-step" algorithm for the modular composition by H, requiring the computation of *only* $\sqrt{N}$ powers of H. The key idea is to see $p(H)$ as a polynomial in $H^{\sqrt{N}}$ of degree $\sqrt{N}$. Computing its coefficients amounts to $\sqrt{N}$ modular compositions of polynomials of degree at most $\sqrt{N}$ by *the same element* H. Brent and Kung (1978, Algorithm 2.1) remarked that these *simultaneous* modular compositions can be performed using $D/\sqrt{N}$ products of pairs of $\sqrt{N} \times \sqrt{N}$ matrices. Adding the $O(\sqrt{N})$ multiplications in Q, the total cost of this algorithm is

$$O\Big(\sqrt{N}\,\mathsf{M}(D) + DN^{(\omega-1)/2}\Big).$$

Tellegen's principle implies that the power projection problem can also be solved within $O\Big(\sqrt{N}\,\mathsf{M}(D) + DN^{(\omega-1)/2}\Big)$ operations in k.

We now make explicit and effectively transpose the maps involved in the algorithm for modular composition described above. In order to simplify notations, set $r = \left\lfloor \sqrt{N} \right\rfloor$ and $G = H^r$. For a polynomial $p = p_0 + \cdots + p_{N-1}T^{N-1}$, set $\widetilde{p_i} = p_{(i-1)r} + \cdots + p_{ir-1}T^{r-1}$. After precomputing the elements $1, H, \ldots, H^r$, Brent and Kung's modular composition algorithm decomposes into three linear maps, as follows:

$$\begin{array}{rcccc} k[T]_{<N} & \to & k[T]_{<r} \times \cdots \times k[T]_{<r} & \to & Q \times \cdots \times Q \quad \to \quad Q \\ p(T) & \mapsto & \Big(\widetilde{p_1}(T), \ldots, \widetilde{p_r}(T)\Big) & & \\ & & (q_1, \ldots, q_r) \quad \mapsto & \Big(q_1(H), \ldots, q_r(H)\Big) & \\ & & & (A_1, \ldots, A_r) \quad \mapsto & \textstyle\sum_{i=1}^{r} A_i G^{i-1} \end{array}$$

- The first one is a splitting map, so that no arithmetic operation is required.
- The second map is $M \mapsto \mathcal{H}M$, where $\mathcal{H}$ is the $r \times D$ matrix whose columns contain the coordinates of the first $r - 1$ powers of H.
- The third map is computed using a Horner-like method.

We now proceed to inspect the transposed maps. Concerning the last one, we first note that, by definition, the transpose of the usual product map $B \mapsto AB$ on Q is the map $\widehat{Q} \to \widehat{Q}$ that associates to $\ell \in \widehat{Q}$ the linear form

$$\begin{aligned} A \circ \ell : Q &\to k \\ B &\mapsto \ell(AB). \end{aligned}$$

We use the classical denomination *transposed product* for the operation $A \circ \ell$. An important property is that it endows the dual $\widehat{Q}$ with a Q-module structure. With this notation, transposing Horner's rule amounts to computing

$G \circ \ell$, then $G \circ (G \circ \ell)$ and so on. The transpose of the second map is simply given by $M \mapsto \mathcal{H}^{\mathrm{tr}} M$ and that of the first map can be computed for free.

In summary, reversing the arrows in the previous diagram gives the following decomposition of the power projection map; recall that $1, H, \ldots, H^{r-1}$ and $G = H^r$ are precomputed.

$$\begin{array}{ccccccc}
k[[T]]_{<N} & \leftarrow k[[T]]_{<r} \times \cdots \times k[[T]]_{<r} \leftarrow & \widehat{Q} \times \cdots \times \widehat{Q} & \leftarrow \widehat{Q} \\
\sum_{i=1}^{r} S_i T^{(i-1)r} & \mapsto & (S_1, \ldots, S_r) & & \\
 & & \left(\sum_{i=0}^{r-1} \ell_j(H^i) T^i \right)_{1 \le j \le r} & \mapsto & (\ell_1, \ldots, \ell_r) & & \\
 & & & & (\ell, \ldots, G^{r-1} \circ \ell) & \mapsto & \ell
\end{array}$$

The corresponding algorithm for the power projection works as described in Figure 7 below; therein we denoted by $M[i, j]$ the (i, j)th entry of a matrix M and by 1_Q the unity of the algebra Q.

Input: H in Q, ℓ in $\widehat{Q}$, $N \geq 1$ in $\mathbb{N}$.
Output: the series $\sum_{i=0}^{N-1} \ell(H^i) T^i + O(T^N)$.

$r \leftarrow \lfloor \sqrt{N} \rfloor, s \leftarrow \lceil N/r \rceil$
$H_0 \leftarrow 1_Q$
`for` p `from` 1 `to` r `do`
$H_p \leftarrow H \cdot H_{p-1}$
$M_1 \leftarrow$ the $D \times r$ matrix whose pth column is H_{p-1}
$\ell_0 \leftarrow \ell$
`for` q `from` 1 `to` $s - 1$ `do`
$\ell_q \leftarrow H_r \circ \ell_{q-1}$
$M_2 \leftarrow$ the $s \times D$ matrix whose qth row is ℓ_{q-1}
$M \leftarrow M_2 M_1$
`return` $\displaystyle\sum_{\substack{1 \le i \le s \\ 1 \le j \le r}} M[i, j] T^{(i-1)r+j-1} + O(T^N)$

Fig. 7. Bivariate power projection

Apart from the computation of a product of two rectangular matrices M_1 and M_2 of sizes $\sqrt{N} \times D$ and $D \times \sqrt{N}$, this algorithm requires $\sqrt{N}$ multiplications in Q and $\sqrt{N}$ transposed products. Decomposing the matrices M_1 and M_2 into $D/\sqrt{N}$ square matrices of size $\sqrt{N}$ allows us to compute their product $M_2 M_1$ within $O\big(DN^{(\omega-1)/2}\big)$ operations in k. In the next sections, we give explicit algorithms for the product and the transposed product in Q, which have complexity $O(\mathsf{M}(D))$. This concludes the proof of Proposition 3.

In contrast to Shoup's (1999) algorithm for the power projection, our algorithm uses fast matrix arithmetic. Interestingly, in (Kaltofen and Shoup, 1998, Algorithm AP), a related question, the *automorphism evaluation problem*, is solved in a similar fashion.

4.3 Representing the linear forms

The quotient algebra Q has a canonical monomial basis: since f has degree m and g has degree n, then

$$\mathcal{M} = \{x^i y^j,\ 0 \leq i \leq m-1, 0 \leq j \leq n-1\}$$

forms a monomial basis of Q, where x and y are the images of X and Y in Q. The linear forms will be given by their coefficients in the dual basis of $\mathcal{M}$, that is, by the list of their values on the elements in $\mathcal{M}$. Then the cost of a single evaluation is $mn = D$ operations in the base field. As a preamble to our algorithm, it is necessary to compute the trace of all elements in the basis $\mathcal{M}$.

Let us thus consider i in $0, \ldots, m-1$ and j in $0, \ldots, n-1$. By Stickelberger's theorem, the trace of $x^i y^j$ equals $\sum_{\alpha,\beta} \alpha^i \beta^j$; then Lemma 3 shows that this trace is the product of the coefficients of T^i in Newton(f) and T^j in Newton(g). The series Newton(f) and Newton(g) can be computed at precision respectively m and n in $O(\mathsf{M}(\max(m,n)))$ base field operations. Then by the above reasoning, the value of the trace form on the canonical basis $\mathcal{M}$ can be computed for $mn = D$ additional multiplications.

4.4 Complexity of the product in Q

Due to the very specific form of the ideal defining our quotient algebra $Q = k[X,Y]/(f(X), g(Y))$, one can design the following algorithm for the product in Q. It takes as input two elements A, B in Q. To obtain AB in Q, we first compute their product as plain polynomials in $k[X,Y]$, then reduce this product modulo $(f(X), g(Y))$.

We use Kronecker's substitution $X \leftarrow T, Y \leftarrow T^{2m-1}$ to reduce the computation of the product of A and B as polynomials in $k[X,Y]$ to a univariate multiplication of polynomials of degree at most $2mn - m - n < 2D$. This can be done in complexity $O(\mathsf{M}(D))$. The resulting product $P = AB$ is a bivariate polynomial of degree at most $2m-2$ in X and at most $2n-2$ in Y. We next reduce it modulo the ideal $(f(X), g(Y))$; this is done in two steps.

We first consider P as a polynomial in $k[X][Y]$ and we reduce all its coefficients

$P_j(X)$ modulo f, using the variant of Sieveking-Kung's algorithm (Sieveking, 1972; Kung, 1974) described in (von zur Gathen and Gerhard, 1999, Algorithm 9.5): given P_j, compute $S_j = \lceil u \cdot \mathrm{rev}(2m-2, P_j) \rceil^{m-1}$, where u denotes the power series $1/\mathrm{rev}(m, f)$ at precision m; then $P_j \mod f$ is given by $P_j - \mathrm{rev}(m-2, S_j) \cdot f$. Besides the precomputation of u, whose cost is $3\mathsf{M}(m) + O(m)$ base field operations, see (von zur Gathen and Gerhard, 1999, Theorem 9.4), this algorithm uses at most $2n(2\mathsf{M}(m) + O(m)) = 4n\mathsf{M}(m) + O(D)$ operations in k.

We have obtained a bivariate polynomial of degree at most $m-1$ in X and at most $2n-2$ in Y, which we now view in $k[Y][X]$. The final step consists in reducing its coefficients modulo $g(Y)$. This is done using again Sieveking-Kung's algorithm, within $2m\mathsf{M}(n) + O(D)$ operations in k, plus the precomputation of $1/\mathrm{rev}(n, g)$, of cost $3\mathsf{M}(n) + O(n)$. As both $m\mathsf{M}(n)$ and $n\mathsf{M}(m)$ are at most $\mathsf{M}(mn) = \mathsf{M}(D)$, our algorithm for the product in Q uses $O(\mathsf{M}(D))$ operations in k. We give the corresponding pseudo-code in Figure 8 below.

Input: $A, B \in Q = k[X,Y]/(f(X), g(Y))$.
Output: the product AB in Q.

$C(T) \leftarrow A(T, T^{2m-1}) \cdot B(T, T^{2m-1})$
$u \leftarrow \lceil 1/\mathrm{rev}(m, f) \rceil^{m-1}$
`for` j `from` 0 `to` $2n-2$ `do`
    $P_j \leftarrow \sum_{i=0}^{2m-2} \mathtt{Coeff}(C, (2m-1)j + i) X^i$
    $S_j \leftarrow \lceil u \cdot \mathrm{rev}(2m-2, P_j) \rceil^{m-1}$
    $P_j \leftarrow P_j - \mathrm{rev}(m-2, S_j) \cdot f$
$v \leftarrow \lceil 1/\mathrm{rev}(n, g) \rceil^{n-1}$
`for` i `from` 0 `to` $m-1$ `do`
    $Q_i \leftarrow \sum_{j=0}^{2n-2} \mathtt{Coeff}(P_j, i) Y^j$
    $R_i \leftarrow \lceil v \cdot \mathrm{rev}(2n-2, Q_i) \rceil^{n-1}$
    $Q_i \leftarrow Q_i - \mathrm{rev}(n-2, R_i) \cdot g$
`return` $\sum_{i=0}^{m-1} Q_i(Y) X^i$

Fig. 8. Bivariate modular multiplication

4.5 Complexity of the transposed product

In this section we propose an algorithm for the transposed product in Q, whose complexity is the same as that of the multiplication in Q. As noticed by Shoup (1994), Proposition 4 already implies that such an algorithm exists, our contribution is to exhibit a simple, ready-to-implement one. We derive it by applying the program transformation techniques introduced by Bostan

et al. (2003a) to the algorithm of Section 4.4. Roughly, this works as follows. Given a program, we decompose it into "elementary" blocks of instructions, then go through from the bottom to the top and *transpose* each block. In this process, the ascending `for` loops are transformed into descending ones, and input and output are swapped. We refer to (Bostan et al., 2003a) for details.

In our case, two main procedures have to be transposed: the bivariate polynomial multiplication using Kronecker's substitution on the one hand, and Sieveking-Kung's algorithm on the other hand. The latter is explicitly transposed by Bostan et al. (2003a). Since Kronecker's substitution is the identity map in the canonical bases, transposing it is immediate, so it remains to transpose the univariate polynomial product involved in the bivariate multiplication. The transposed map of the multiplication by a fixed polynomial P of degree m is the *middle product* defined by Hanrot et al. (2004): it sends Q of degree at most $m+n$ to the polynomial $[\mathrm{rev}(m,P)\cdot Q]_m^{m+n+1}$, denoted $\mathtt{mul}^t(n,P,Q)$. By Proposition 4 it can be computed for the cost of one multiplication of two polynomials of degree m and n, up to $O(m)$ operations, see (Hanrot et al., 2004; Bostan et al., 2003a) for explicit algorithms. Note that the transposed Sieveking-Kung's algorithm in Bostan et al. (2003a) also uses middle products. The corresponding algorithm goes as in Figure 9.

Input: A in $Q = k[X,Y]/(f(X),g(Y))$, ℓ in $\widehat{Q}$.
Output: the transposed product $A \circ \ell$.

$v \leftarrow \lceil 1/\mathrm{rev}(n,g) \rceil^{n-1}$
`for` i `from` $m-1$ `downto` 0 `do`
  $Q_i \leftarrow \sum_{j=0}^{n-1} \ell(x^i y^j) Y^j$
  $R_i \leftarrow \mathtt{mul}^t(n-2, g, Q_i)$
  $Q_i \leftarrow Q_i - X^n \lceil R_i \cdot v \rceil^{n-1}$
$u \leftarrow \lceil 1/\mathrm{rev}(m,f) \rceil^{m-1}$
`for` j `from` $2n-2$ `downto` 0 `do`
  $P_j \leftarrow \sum_{i=0}^{m-1} \mathtt{Coeff}(Q_i, j) X^i$
  $S_j \leftarrow \mathtt{mul}^t(m-2, f, P_j)$
  $P_j \leftarrow P_j - X^m \lceil S_j \cdot u \rceil^{m-1}$
$P(X,Y) \leftarrow \sum_{j=0}^{2n-2} P_j(X) Y^j$
$C \leftarrow \mathtt{mul}^t\big(2mn-m-n, A(T,T^{2m-1}), P(T,T^{2m-1})\big)$
`return` $\displaystyle\sum_{\substack{0\le i\le m-1\\ 0\le j\le n-1}} \mathtt{Coeff}(C, (2m-1)j+i) X^i Y^j$

Fig. 9. Bivariate transposed product

This program has been constructed by operating some transformation on the instructions of the program for the dual question in Section 4.4. Its correctness is guaranteed by the validity of these transformations techniques. However, we

conclude this section by interpreting what is computed.

The algorithm takes as input a linear form ℓ in $\widehat{Q}$. Since $f(x) = 0$ in Q, for any integer $j \geq 0$, the sequence $\ell(x^i y^j)_{i \geq 0}$ satisfies a linear recurrence with constant coefficients, of characteristic polynomial f. The problem of extending such a linear recurrent sequence is dual to the division with remainder by f, see (Bostan et al., 2003a, Section 5), so in the first `for` loop, the algorithm computes the values $\ell(x^i y^j)$, for $0 \leq j \leq n-1$ and $m \leq i \leq 2m-2$. Similarly, after the pass through the second `for` loop, all the values taken by ℓ on the monomials in the set $\mathcal{M}^2 = \{x^i y^j, 0 \leq i \leq 2m-2, 0 \leq j \leq 2n-2\}$ are computed; these values are encoded in the polynomial $P(X, Y)$. By definition of the middle product, one can finally check that the algorithm outputs the part supported by $\mathcal{M}$ in the product $A(\frac{1}{X}, \frac{1}{Y}) \cdot P(X, Y)$, that is, in the product

$$A\Big(\frac{1}{X}, \frac{1}{Y}\Big) \cdot \sum_{x^i y^j \in \mathcal{M}^2} \ell(x^i y^j) X^i Y^j.$$

Proposition 1 in (Bostan et al., 2003b) shows that this part gives the coefficients of $A \circ \ell$ in the dual basis of $\mathcal{M}$ and this finishes an alternative proof of the correctness of our algorithm.

4.6 Experimental results

We implemented our diamond product algorithm in the NTL C++ library (Shoup, 1996–2005); we implemented the version for large or zero characteristic, for base fields of type $\mathbb{Z}/p\mathbb{Z}$, with p prime. Our implementation uses Winograd's variant of Strassen's (1969) matrix multiplication algorithm; it also uses the implementation of transposed polynomial multiplication developed by Bostan et al. (2003a).

The data used to test the diamond product are similar to those of Section 3.4: the input polynomials $f(X)$ and $g(Y)$ have equal degrees $m = n$ and their coefficients are chosen uniformly at random; the output has degree $D = m^2$. The polynomial H is randomly chosen of degree less than m in both X and Y. We let m vary from 1 to 425; the base field is defined by a 32 bit prime.

Figure 10 shows the time for the diamond product computation, using both classical and Strassen's matrix multiplication. Again, the abrupt time jumps that occur at powers of 2 come from the Fourier transform. In Figure 11, we separate the times used in the polynomial multiplication and transposed multiplication step, on the one hand, and the linear algebra step, using respectively classical and Strassen multiplication, on the other hand. The polynomial multiplication step is predominant, but the ratio between this step and the linear algebra one actually reduces as the degree grows.

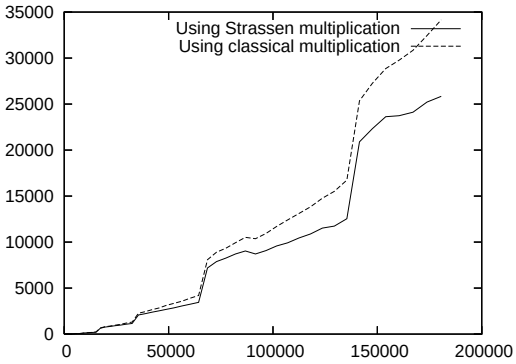

Fig. 10. Diamond product.
(Time in sec. vs output degree)

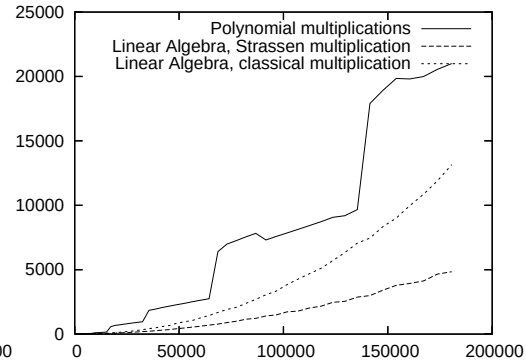

Fig. 11. Respective times for polynomial multiplications & linear algebra by classical and Strassen's algorithm.
(Time in sec. vs output degree)

Recall that the diamond product algorithm handles matrices whose size $m = \sqrt{D}$ varies from 1 to 425. It appears that for such sizes, using a fast matrix multiplication algorithm does have a practical significance on the whole diamond product computation time. Indeed we save a factor of up to 3 on the linear algebra phase, and more than 25% on the whole computation time.

5 Applications and Related Questions

To conclude this article, we present situations where composed operations, notably sums and products, are useful.

Algebraic numbers. One may represent an algebraic number by its minimal polynomial. If α and β are two algebraic numbers represented by their minimal polynomials $f(x)$ and $g(y)$, the sum $\alpha+\beta$ is represented by one of the irreducible factors of the composed sum $f \oplus g$. The product $\alpha\beta$ is represented by one of the irreducible factors of the composed product $f \otimes g$ (subtractions $\alpha-\beta$ and divisions α/β can be handled similarly). Thus the resultant methods described in (Loos, 1983; Cohen, 1993) can be replaced by our faster solutions, even though factoring the output remains necessary and is possibly costly.

We mention that Kaltofen (2000) presents an alternative solution for this question: it consists in factoring f in the algebraic extension $\mathbb{Q}[y]/(g(y))$ beforehand, and then computing a power projection modulo a system of the form $g(y) = 0, h(x, y) = 0$. Thus, the factorization in degree $\deg(f)\deg(g)$ over $\mathbb{Q}$ is replaced by a factorization in degree $\deg(f)$ in a number field of degree $\deg(g)$.

Algebraic functions. Our algorithms also adapt to operations over Puiseux series as it suffices to operate with a base field of the form $k(z)$. As a matter

of fact, ideas similar to the ones developed in Section 3 prove useful in determining the generating function of walks over the half-line determined by a fixed finite set of allowed jumps; see the "Platypus Algorithm" [sic] and the discussion of (Banderier and Flajolet, 2002, pp. 56–58). (There the problem is to calculate the minimal polynomial satisfied by a product $\alpha_1 \cdots \alpha_k$ of k distinct branches of an algebraic function defined by $P(z, \alpha) = 0$.)

Dispersion set of polynomials. In many algorithms for symbolic summation (e.g., Abramov, 1971; Gosper, 1978; Abramov, 1989; Petkovšek, 1992; Paule, 1995) one has to compute *the dispersion set* of two polynomials, that is, the set of the (integer) distances between their roots. Classically, this is done by computing the polynomial whose roots are the elements of the dispersion set. The latter polynomial is again a resultant of the particular form discussed in Section 3 and can be computed fast using our algorithms. Alternative methods for determining the dispersion set have been designed by Man and Wright (1994); Gerhard et al. (2003). It would be interesting to carefully compare the (bit) complexities and the practical performances of these algorithms.

Irreducible polynomials. Constructing irreducible polynomials of prescribed degree over finite fields is a useful but difficult task. It serves, for instance, to implement arithmetic in extension fields. The most efficient algorithm known is due to Shoup (1990, 1994): it consists in first constructing irreducible polynomials whose degree is a prime power, then combining them by means of composed products. In (Shoup, 1994), the second step is achieved by a minimal polynomial computation, which has complexity $O\big(D^{(\omega+1)/2}\big)$, where D is the output degree. Thanks to our algorithm for the composed product, the cost of this step becomes linear in D, up to logarithmic factors.

Point counting. Designing genus 2 hyperelliptic cryptosystems requires determining the cardinality of the Jacobian of genus 2 curves defined over finite fields. When the base field is a prime field of the form $\mathbb{Z}/p\mathbb{Z}$, a commonly used solution is the extension by Gaudry and Harley (2000) of Schoof's (1985) algorithm for elliptic curves, which requires to compute torsion subgroups of the Jacobian.

Working out the details, one is led to solve systems of the form

$$\frac{f(x_1)}{g(x_1)} = \frac{f(x_2)}{g(x_2)}, \quad \frac{h(x_1)}{g(x_1)} = \frac{h(x_2)}{g(x_2)},$$

where f, g, h are univariate polynomials. Taking into account the symmetry in x_1, x_2, we wish to compute an eliminating polynomial for $x_1 + x_2$. This can be done through a suitable resultant computation, but the denominators $g(x_1)$ and $g(x_2)$ create high-degree spurious factors in this resultant, which should be

found and removed. The parasites are powers of the composed sum of g with itself; in their cryptographic-size record, Gaudry and Schost (2004) encounter degrees in the range of several hundreds of thousands. To treat problems of such sizes, the use of our fast algorithms for composed sums becomes necessary.

Linear recurrent sequences with infinitely many zeros. A classical result (Berstel and Mignotte, 1976) asserts that a linear recurrent sequence has infinitely many zero terms if its minimal polynomial f has a unitary pair, that is, if it has two roots whose ratio is a root of unity. Yokoyama et al. (1995) give algorithms to test this condition, and if so, to find the order of the multiplicative group generated by the corresponding roots of unity. The most time-consuming part of their algorithm is the computation of a polynomial whose roots are the ratios of all pairs of roots of f. This directly reduces to the computation of a composed product, for which our algorithms apply.

Shift of polynomials. In (von zur Gathen and Gerhard, 1997), six algorithms for computing shifts of polynomials are proposed and their complexity is analyzed. A seventh algorithm can be deduced as a straightforward application of our algorithm for the composed sums, since $f \oplus (T + a)$ is the shift polynomial of f by a. In characteristic zero, the complexity of this algorithm is linear (up to logarithmic factors) in the degree of f, in terms of base field operations. Yet, the convolution method of Aho et al. (1975) is better by a constant factor. In small characteristic, the analysis has yet to be done.

Construction of polynomials that are hard to factor. Our algorithms for composed sums can be used to compute the Swinnerton-Dyer polynomials (Zippel, 1993, p. 305), defined as $\prod (T \pm \sqrt{p_1} \pm \cdots \pm \sqrt{p_n})$, where the product runs over all 2^n possible combinations of $\pm$ signs and $p_1, \ldots, p_n$ are distinct primes. They also apply to the computation of their bivariate analogues (Zippel, 1993, p. 340), which are the bivariate polynomials of total degree $D = 2^n$ defined by $S_n(X, Y) = \prod (Y \pm \sqrt{X + 1} \pm \cdots \pm \sqrt{X + n})$. (These families of polynomials are known to exhibit the worst possible case for factorization algorithms, over $\mathbb{Z}[T]$ and $k[X, Y]$, based on the "lift and recombine" strategy.) Without getting into details, an evaluation/interpolation scheme together with our algorithms for composed sums allow the computation of $S_n(X, Y)$ in $O(D\,\mathsf{M}(D))$ operations in k, which is again nearly optimal in the size of the output. Similarly, one can compute all the "hard to factor" polynomials in (Kaltofen et al., 1983). Moreover, performing composed sums of polynomials of Swinnerton-Dyer type allows one to build polynomials of prescribed degrees which are irreducible in $\mathbb{Z}[T]$, but reducible modulo any prime p. This yields a constructive proof of the result in (Brandl, 1986), in the case of nonsquarefree degrees.

Resolvents. Resolvents are an important tool in Galois theory, notably for the *direct problem* of determining the Galois group of an irreducible polynomial f of degree m. Their factorization patterns help determine the Galois group of f. For $h \leq m$, an example of such a resolvent is the polynomial f^{+h} of degree $N = \binom{m}{h}$, whose roots are the sums $\alpha_{i_1} + \cdots + \alpha_{i_h}$, with $1 \leq i_1 < \cdots < i_h \leq m$, where $(\alpha_i)_{1 \leq i \leq m}$ are the roots of f. This differs from the hth iterated composed sum, since repetitions of roots are not allowed here. Yet, the methods we have presented help answer some simple cases, as illustrated in the next example.

Let $f(T) = T^7 - 7T + 3$ be the Cartier polynomial introduced by Giusti et al. (1989), and $F = f^{+3}$ the polynomial whose roots are all sums of $h = 3$ distinct roots of f. To prove that the Galois group of f is not the symmetric group $\mathcal{S}_7$, it is enough to check that F is not irreducible. The polynomial F has degree 35, so knowing its Newton series to order 35 suffices to recover it. To do this, we first decompose $f^{\oplus 3} = f \oplus f \oplus f$ as

$$f^{\oplus 3} = \prod_{\alpha} (T - 3\alpha) \quad \cdot \prod_{\alpha \neq \beta} \Big(T - (\alpha + 2\beta)\Big)^3 \cdot \prod_{\alpha \neq \beta \neq \gamma \neq \alpha} \Big(T - (\alpha + \beta + \gamma)\Big)^6.$$

Then, using the definition $F = \prod_{\alpha \neq \beta \neq \gamma \neq \alpha} \Big(T - (\alpha + \beta + \gamma)\Big)$ and the equalities

$$\prod_{\alpha} (T - 3\alpha) = f \otimes (T - 3) \text{ and } \prod_{\alpha \neq \beta} \Big(T - (\alpha + 2\beta)\Big) = \frac{f \oplus \Big(f \otimes (T-2)\Big)}{f \otimes (T-3)}$$

enables us to express $\mathrm{Newton}(F)$ as

$$\frac{1}{6}\Big(\mathrm{Newton}\Big(f^{\oplus 3}\Big) + 2\ \mathrm{Newton}\Big(f \otimes (T-3)\Big) - 3\ \mathrm{Newton}\Big(f \oplus \Big(f \otimes (T-2)\Big)\Big)\Big).$$

Using Lemmas 3 and 4, this series can be computed from the series $\mathrm{Newton}(f)$ and $\exp(T)$ to order 35. The polynomial F is then recovered using the algorithms in Section 2. The CPU time used in a direct resultant computation is about 300 times the whole computation time using our approach.

A straightforward generalization of this approach for an arbitrary h is not satisfactory, due to the combinatorial explosion of the number of terms involved. A faster method is presented by Casperson and McKay (1994) and has complexity $\widetilde{O}\left(h^2 N + N^2\right)$. It is based on the following recurrence relation, expressing f^{+h} in terms of f^{+j}, for $j < h$:

$$\Big(f^{+h}\Big)^h = \prod_{i=1}^{h} \Big(\Big(f \otimes (T - i)\Big) \oplus f^{+(h-i)}\Big)^{(-1)^{i+1}}.$$

Using this formula and the fast conversion algorithms presented in Section 2 reduces the complexity to $\widetilde{O}\left(hN\right)$. Nevertheless, the degree of the output is N, so an optimal algorithm for this question has yet to be found.

Characteristic polynomials in univariate quotient algebras. By definition, the diamond product includes as a very particular case the characteristic polynomial of a univariate polynomial $H(X)$ in an univariate quotient algebra $k[X]/(b(X))$. The latter can thus be computed using $O(\sqrt{n}(\mathsf{M}(n)+n^{\omega/2}))$ operations in k, where $n = \deg(b) > \deg(H)$. The resulting algorithm is similar to that for minimal polynomials in (Shoup, 1999).

Rothstein-Trager resultants. A special type of bivariate resultant occurs in algorithms for symbolic integration, see (Trager, 1976; Rothstein, 1977). Given $a(X)$ and $b(X)$ in $k[X]$ of degree at most n, with b squarefree, these algorithms require the computation of $r(Y) = \mathrm{Res}_X(Yb'(X) - a(X), b(X))$, which has degree at most n. A direct evaluation-interpolation based algorithm has complexity $O(n\,\mathsf{M}(n)\log(n))$. A different approach is based on the fact that $r(X)$ equals, up to the factor $\mathrm{Res}(b, b')$, the characteristic polynomial of $H(X) = a/b' \bmod b$ in $k[X]/(b)$, thus can be computed as explained in the preceding paragraph.

Graeffe polynomials. Let f be a monic polynomial of degree m and N be a positive integer. We call Nth *Graeffe polynomial* of f the polynomial of degree m whose roots are the Nth powers of the roots of f.

This polynomial can be obtained using $O(\mathsf{M}(mN))$ operations in k, by computing the composed product of f and $X^N - 1$. Note that the same complexity result is announced in (Henrici, 1986, Section 13.8). This is nearly optimal with respect to m, but not to N. On the other hand, the Nth Graeffe polynomial of f is the characteristic polynomial of X^N modulo f. Computing $X^N \bmod f$ has complexity $O(\mathsf{M}(m)\log(N))$, which is optimal in N, but then the characteristic polynomial computation has complexity more than linear in m.

For comparison, Graeffe polynomials are computed by von Haeseler and Jürgensen (2001) by means of so-called *decimation matrices*, which are structured (Toeplitz, quasi-circulant) $N \times N$ matrices with polynomial entries of degree at most $\frac{m+N}{N}$. Using an evaluation-interpolation scheme, the cost of this method is dominated by the evaluation of m determinants of $N \times N$ scalar Toeplitz matrices. Over $\mathbb{C}$, Toeplitz determinants can be evaluated in $O(\mathsf{M}(N)\log N)$ operations, using the algorithm in (Kravanja and Van Barel, 2000).

Is there a way of reducing the cost to $O(\mathsf{M}(m)\log(N))$? If N is a power of 2, this can be achieved using binary powering, but the general case remains open.

Acknowledgments. We wish to thank the referees of an earlier version of this article for their helpful scholarly comments. This work has been partly supported by the European Commission under the Future and Emerging Technologies program of the Fifth Framework, ALCOM-FT project IST-1999-14186.

References

Abramov, S. A., 1971. On the summation of rational functions. Akademiya Nauk SSSR. Zhurnal Vychislitel′noĭ Matematiki i Matematicheskoĭ Fiziki 11 (4), 1071–1075, english translation in U.S.S.R. Computational Mathematics and Mathematical Physics, 324–330.

Abramov, S. A., 1989. Rational solutions of linear differential and difference equations with polynomial coefficients. Akademiya Nauk SSSR. Zhurnal Vychislitel′noĭ Matematiki i Matematicheskoĭ Fiziki 29 (11), 1611–1620, 1757, english translation in U.S.S.R. Computational Mathematics and Mathematical Physics, 7–12.

Aho, A. V., Steiglitz, K., Ullman, J. D., 1975. Evaluating polynomials at fixed sets of points. SIAM Journal of Computing 4 (4), 533–539.

Banderier, C., Flajolet, P., 2002. Basic analytic combinatorics of directed lattice paths. Theoretical Computer Science 281 (1-2), 37–80.

Berstel, J., Mignotte, M., 1976. Deux propriétés décidables des suites récurrentes linéaires. Bulletin de la Société Mathématique de France 104 (2), 175–184.

Bini, D., Pan, V. Y., 1994. Polynomial and matrix computations. Vol. 1. Birkhäuser Boston Inc., Boston, MA.

Bostan, A., Flajolet, P., Salvy, B., Schost, É., Oct. 2002. Fast computation with two algebraic numbers. Research Report 4579, Institut National de Recherche en Informatique et en Automatique, 20 pages.

Bostan, A., González-Vega, L., Perdry, H., Schost, É., 2005. From Newton sums to coefficients: complexity issues in characteristic p. In: MEGA'05. To appear.

Bostan, A., Lecerf, G., Schost, É., 2003a. Tellegen's principle into practice. In: ISSAC'03. ACM Press, pp. 37–44.

Bostan, A., Salvy, B., Schost, É., 2003b. Fast algorithms for zero-dimensional polynomial systems using duality. Applicable Algebra in Engineering, Communication and Computing 14 (4), 239–272.

Brandl, R., 1986. Integer polynomials that are reducible modulo all primes. The American Mathematical Monthly 93 (4), 286–288.

Brawley, J. V., Carlitz, L., 1987. Irreducibles and the composed product for polynomials over a finite field. Discrete Mathematics 65 (2), 115–139.

Brawley, J. V., Gao, S., Mills, D., 1999. Computing composed products of polynomials. In: Finite fields: theory, applications, and algorithms (Waterloo, ON, 1997). Vol. 225 of Contemp. Math. AMS, pp. 1–15.

Brent, R. P., 1976. Multiple-precision zero-finding methods and the complexity of elementary function evaluation. In: Analytic computational complexity (Pittsburgh, 1975). Academic Press, pp. 151–176.

Brent, R. P., Kung, H. T., 1978. Fast algorithms for manipulating formal power series. Journal of the Association for Computing Machinery 25 (4), 581–595.

Briand, E., González-Vega, L., 2002. Multivariate Newton sums: identities and

generating functions. Communications in Algebra 30 (9), 4527–4547.
Bürgisser, P., Clausen, M., Shokrollahi, M. A., 1997. Algebraic complexity theory. Vol. 315 of Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]. Springer-Verlag, Berlin.
Cantor, D. G., Kaltofen, E., 1991. On fast multiplication of polynomials over arbitrary algebras. Acta Informatica 28 (7), 693–701.
Casperson, D., McKay, J., 1994. Symmetric functions, m-sets, and Galois groups. Mathematics of Computation 63 (208), 749–757.
Cohen, H., 1993. A course in computational algebraic number theory. Vol. 138 of Graduate Texts in Mathematics. Springer-Verlag, Berlin.
Coppersmith, D., Winograd, S., Mar. 1990. Matrix multiplication via arithmetic progressions. Journal of Symbolic Computation 9 (3), 251–280.
Cox, D., Little, J., O'Shea, D., 1998. Using algebraic geometry. Vol. 185 of Graduate Texts in Mathematics. Springer-Verlag, New York.
Dvornicich, R., Traverso, C., 1989. Newton symmetric functions and the arithmetic of algebraically closed fields. In: AAECC-5 (Menorca, 1987). Vol. 356 of LNCS. Springer, pp. 216–224.
von zur Gathen, J., Gerhard, J., 1997. Fast algorithms for Taylor shifts and certain difference equations. In: ISSAC'97. ACM Press, pp. 40–47.
von zur Gathen, J., Gerhard, J., 1999. Modern computer algebra. Cambridge University Press, New York.
Gaudry, P., Harley, R., 2000. Counting points on hyperelliptic curves over finite fields. In: Proceedings of ANTS IV. Vol. 1838 of LNCS. Springer-Verlag, pp. 313–332.
Gaudry, P., Schost, É., 2004. Construction of secure random curves of genus 2 over prime fields. In: Advances in Cryptology – EUROCRYPT 2004. Vol. 3027 of LNCS. Springer-Verlag, pp. 239–256.
Gerhard, J., Giesbrecht, M., Storjohann, A., Zima, E. V., 2003. Shiftless decomposition and polynomial-time rational summation. In: ISSAC'03. ACM Press, pp. 119–126.
Giusti, M., Lazard, D., Valibouze, A., 1989. Algebraic transformations of polynomial equations, symmetric polynomials and elimination. In: ISSAC'88. Vol. 358 of LNCS. Springer-Verlag, pp. 309–314.
González-López, M.-J., González-Vega, L., 1998. Newton identities in the multivariate case: Pham systems. In: Gröbner bases and applications (Linz, 1998). Vol. 251 of London Mathematical Society Lecture Note Series. Cambridge University Press, pp. 351–366.
González-Vega, L., Trujillo, G., 1995a. Implicitization of parametric curves and surfaces by using symmetric functions. In: ISSAC'95. ACM Press, pp. 180–186.
González-Vega, L., Trujillo, G., 1995b. Using symmetric functions to describe the solution set of a zero-dimensional ideal. In: AAECC-11 (Paris, 1995). Vol. 948 of LNCS. Springer, pp. 232–247.
Gosper, Jr., R. W., 1978. Decision procedure for indefinite hypergeometric summation. Proceedings of the National Academy of Sciences of the United

States of America 75 (1), 40–42.
von Haeseler, F., Jürgensen, W., 2001. Irreducible polynomials generated by decimations. In: Finite fields and their applications (Augsburg, 1999). Springer, pp. 224–231.
Hanrot, G., Quercia, M., Zimmermann, P., 2004. The middle product algorithm, I. Speeding up the division and square root of power series. Applicable Algebra in Engineering, Communication and Computing 14 (6), 415–438.
Henrici, P., 1986. Applied and computational complex analysis. Vol. 3. John Wiley & Sons Inc., New York.
van Hoeij, M., 2002. Factoring polynomials and the knapsack problem. Journal of Number Theory 95 (2), 167–189.
Huang, X., Pan, V. Y., Jun 1998. Fast rectangular matrix multiplication and applications. Journal of Complexity 14 (2), 257–299.
Kaltofen, E., 2000. Challenges of symbolic computation: my favorite open problems. Journal of Symbolic Computation 29 (6), 891–919.
Kaltofen, E., Musser, D. R., Saunders, B. D., 1983. A generalized class of polynomials that are hard to factor. SIAM Journal on Computing 12 (3), 473–483.
Kaltofen, E., Pan, V., 1992. Processor-efficient parallel solution of linear systems II: the positive characteristic and singular cases. In: FOCS'92. IEEE Computer Society Press, pp. 714–723.
Kaltofen, E., Pan, V. Y., 1994. Parallel solution of Toeplitz and Toeplitz-like linear systems over fields of small positive characteristic. In: PASCO '94. Vol. 5 of Lecture Notes Ser. Comput. World Sci. Publishing, pp. 225–233.
Kaltofen, E., Shoup, V., 1998. Subquadratic-time factoring of polynomials over finite fields. Mathematics of Computation 67 (223), 1179–1197.
Kravanja, P., Van Barel, M., 2000. Coupled Vandermonde matrices and the superfast computation of Toeplitz determinants. Numerical Algorithms 24 (1-2), 99–116.
Kronecker, L., 1882. Grundzüge einer arithmetischen Theorie der algebraischen Grössen. Journal für die reine und angewandte Mathematik 92, 1–122.
Kung, H. T., 1974. On computing reciprocals of power series. Numerische Mathematik 22, 341–348.
Lascoux, A., 1986. La résultante de deux polynômes. In: Séminaire d'algèbre P. Dubreil et M.-P. Malliavin. Vol. 1220 of LNM. Springer, pp. 56–72.
Le Verrier, U., 1840. Sur les variations séculaires des éléments elliptiques des sept planètes principales : Mercure, Vénus, La Terre, Mars, Jupiter, Saturne et Uranus. Journal de Mathématiques Pures et Appliquées 4, 220–254.
Lickteig, T., Roy, M.-F., 1996. Cauchy index computation. Calcolo 33 (3-4), 337–351.
Lickteig, T., Roy, M.-F., 2001. Sylvester-Habicht sequences and fast Cauchy index computation. Journal of Symbolic Computation 31 (3), 315–341.
Loos, R., 1983. Computing in algebraic extensions. In: Computer algebra. Springer, pp. 173–187.
Man, Y.-K., Wright, F., 1994. Fast polynomial dispersion computation and its

application to indefinite summation. In: ISSAC'94. ACM Press, pp. 175–180.
Pan, V. Y., 1996. Parallel computation of polynomial GCD and some related parallel computations over abstract fields. Theoretical Computer Science 162 (2), 173–223.
Pan, V. Y., 1997. Faster solution of the key equation for decoding BCH error-corecting codes. In: Proceedings STOC'97. ACM Press, pp. 168–175.
Pan, V. Y., 2000. New techniques for the computation of linear recurrence coefficients. Finite Fields and their Applications 6 (1), 93–118.
Paterson, M. S., Stockmeyer, L. J., Mar. 1973. On the number of nonscalar multiplications necessary to evaluate polynomials. SIAM Journal on Computing 2 (1), 60–66.
Paule, P., 1995. Greatest factorial factorization and symbolic summation. Journal of Symbolic Computation 20 (3), 235–268.
Petkovšek, M., 1992. Hypergeometric solutions of linear recurrences with polynomial coefficients. Journal of Symbolic Computation 14 (2-3), 243–264.
Reischert, D., 1997. Asymptotically fast computation of subresultants. In: ISSAC'97. ACM Press, pp. 233–240.
Rifà, J., Borrell, J., 1991. Improving the time complexity of the computation of irreducible and primitive polynomials in finite fields. In: AAECC-9 (New Orleans, 1991). Vol. 539. pp. 352–359.
Rothstein, M., 1977. A new algorithm for the integration of exponential and logarithmic functions. In: Proceedings of the 1977 MACSYMA Users Conference. NASA Pub. CP-2012, pp. 263–274.
Rouillier, F., 1999. Solving zero-dimensional systems through the Rational Univariate Representation. Applicable Algebra in Engineering, Communication and Computing 9 (5), 433–461.
Schönhage, A., 1977. Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2. Acta Informatica 7, 395–398.
Schönhage, A., 1982. The fundamental theorem of algebra in terms of computational complexity. Tech. rep., Univ. Tübingen, 73 pages.
Schönhage, A., 1993. Fast parallel computation of characteristic polynomials by Leverrier's power sum method adapted to fields of finite characteristic. In: Automata, languages and programming (Lund, 1993). Vol. 700 of LNCS. Springer, pp. 410–417.
Schönhage, A., Strassen, V., 1971. Schnelle Multiplikation großer Zahlen. Computing 7, 281–292.
Schoof, R., 1985. Elliptic curves over finite fields and the computation of square roots mod p. Mathematics of Computation 44, 483–494.
Schost, É., 2005. Multivariate power series multiplication. In: ISSAC'05. To appear.
Schwartz, J. T., 1980. Fast probabilistic algorithms for verification of polynomial identities. Journal of the Association for Computing Machinery 27 (4), 701–717.
Shoup, V., 1990. New algorithms for finding irreducible polynomials over finite fields. Mathematics of Computation 54 (189), 435–447.

Shoup, V., 1994. Fast construction of irreducible polynomials over finite fields. Journal of Symbolic Computation 17 (5), 371–391.

Shoup, V., 1995. A new polynomial factorization algorithm and its implementation. Journal of Symbolic Computation 20 (4), 363–397.

Shoup, V., 1996–2005. NTL: A library for doing number theory. `http://www.shoup.net`.

Shoup, V., 1999. Efficient computation of minimal polynomials in algebraic extensions of finite fields. In: ISSAC'99. ACM Press, pp. 53–58.

Sieveking, M., 1972. An algorithm for division of powerseries. Computing 10, 153–156.

Strassen, V., 1969. Gaussian elimination is not optimal. Numerische Mathematik 13, 354–356.

Thiong Ly, J.-A., 1989. Note for computing the minimum polynomial of elements in large finite fields. In: Coding theory and applications (Toulon, 1988). Vol. 388 of LNCS. Springer, pp. 185–192.

Trager, B. M., 1976. Algebraic factoring and rational function integration. In: SYMSAC'76. ACM Press, pp. 219–226.

Valibouze, A., 1989. Fonctions symétriques et changements de bases. In: EUROCAL '87 (Leipzig, 1987). Vol. 378 of LNCS. Springer, pp. 323–332.

Yokoyama, K., Li, Z., Nemes, I., 1995. Finding roots of unity among quotients of the roots of an integral polynomial. In: ISSAC'95. ACM Press, pp. 85–89.

Zippel, R., 1993. Effective Polynomial Computation. Kluwer Academic Publishers, Boston.